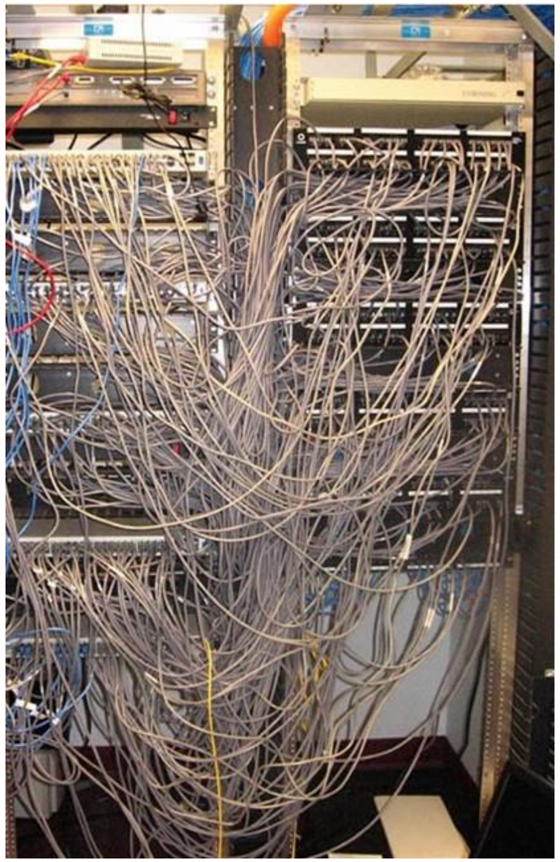
PEMROGRAMAN LANJUT

Code Convention

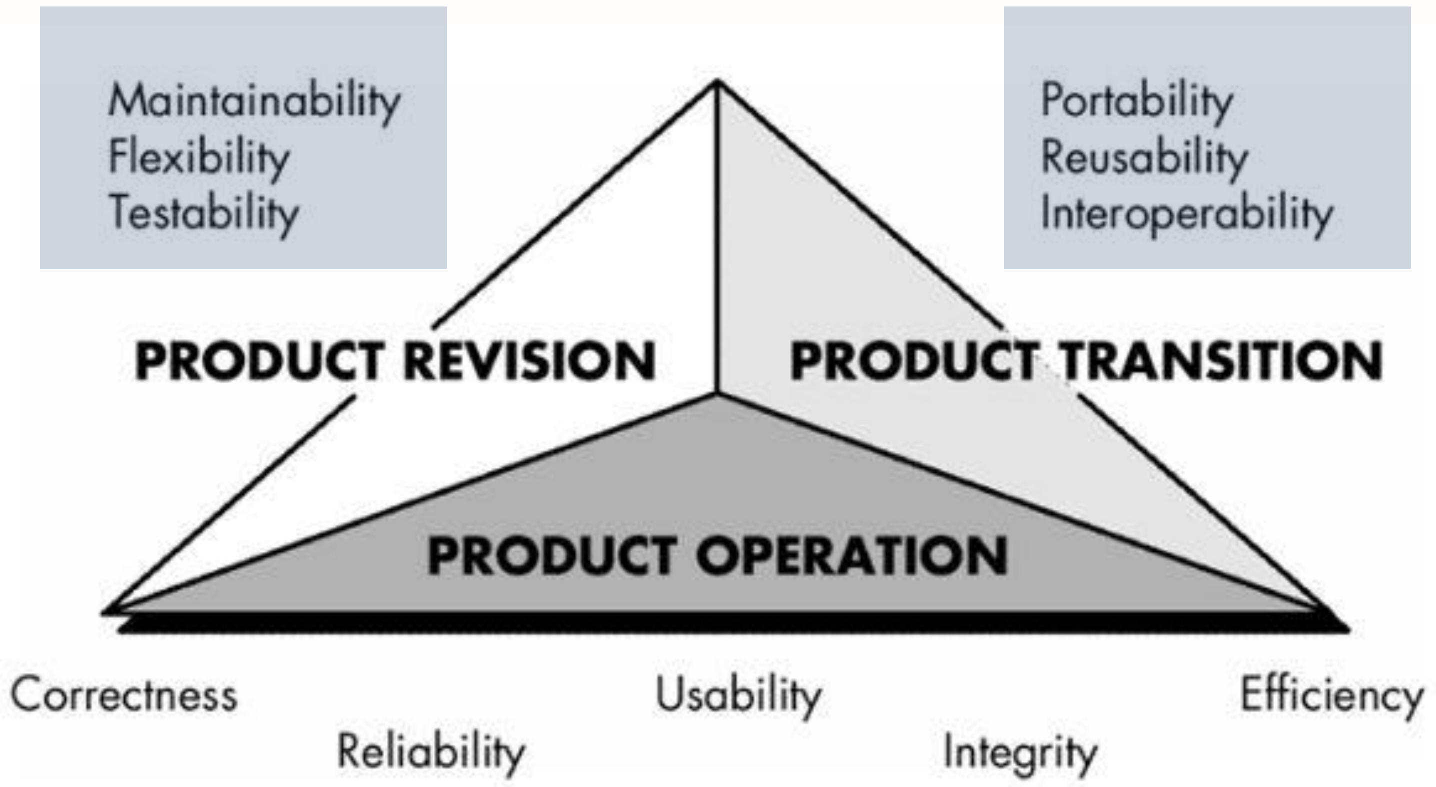
Oleh Politeknik Elektronika Negeri Surabaya
2021

Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer



Bad Code

Clean Code



Mc Call Software Quality Metric

Code Conventions

Set of **guidelines** for a specific programming language that **recommend programming style, practices, and methods** for **each aspect** of a program written in that language.

Code Convention References

Android and Java

1. https://source.android.com/setup/contribute/code-style
2. https://firefox-source-docs.mozilla.org/code-quality/coding-style/index.html
3. https://github.com/ribot/android-guidelines
4. https://www.oracle.com/java/technologies/javase/codeconventions-contents.html

Python

1. https://www.python.org/dev/peps/pep-0008/

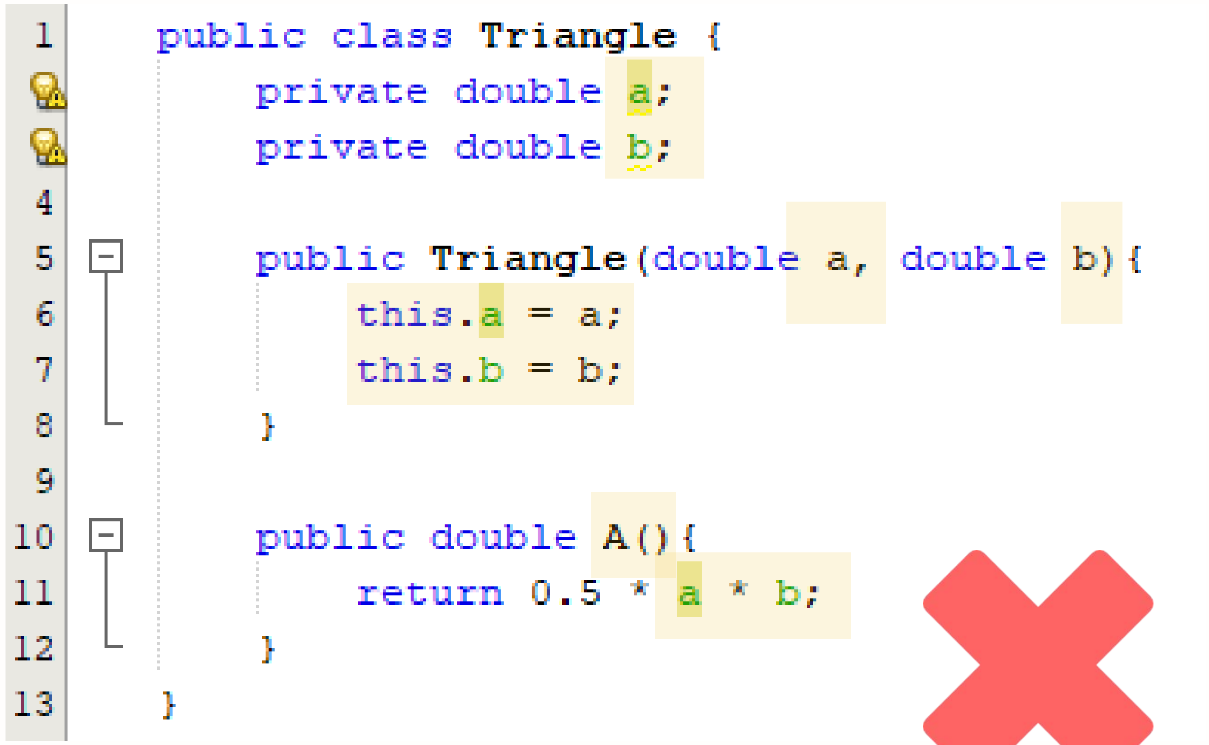
PHP

1. https://www.php-fig.org/psr/psr-1/

Naming Convention: Use Intention-revealing Name

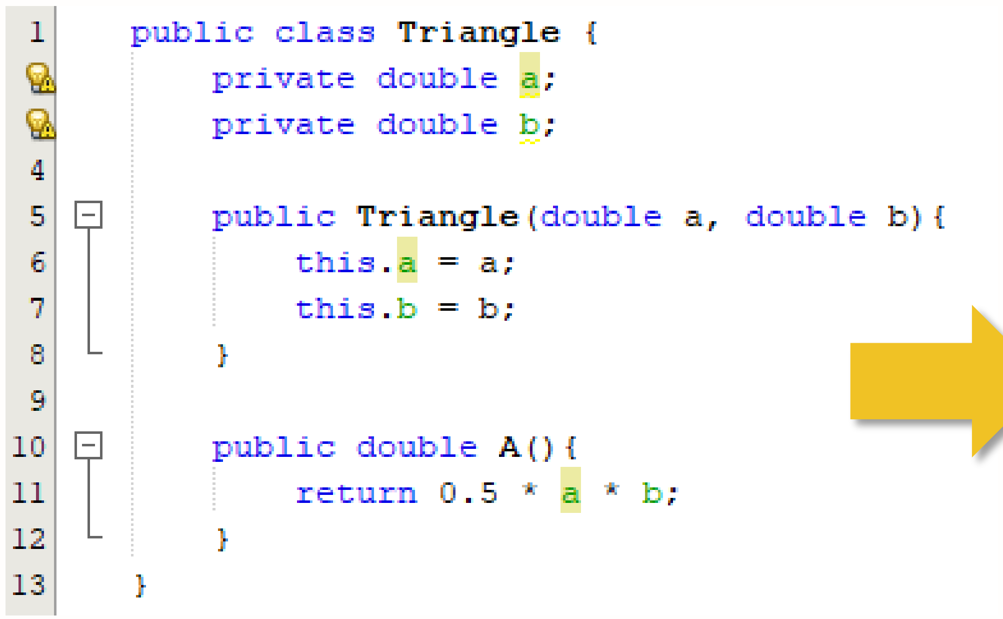
- It should tell you why it exists, what it does, and how it is used.
- If a name requires a comment, then the name does not reveal its intent.

Naming Convention: Use Intention-revealing Name

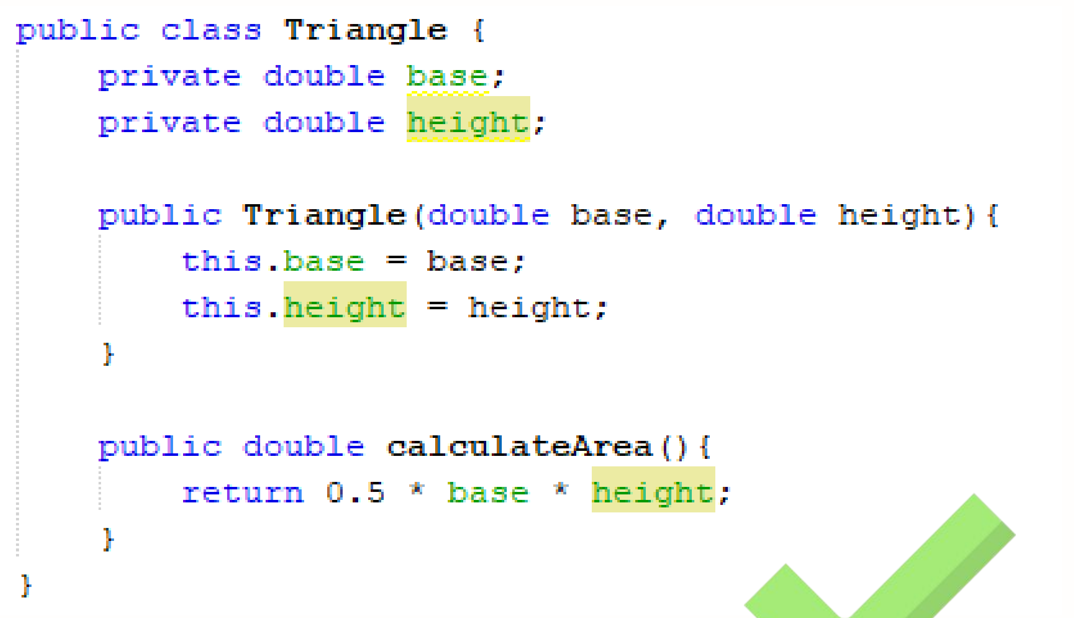


```
public class Triangle {
    private double a;
    private double b;

    public Triangle(double a, double b){
        this.a = a;
        this.b = b;
    }

    public double A(){
        return 0.5 * a * b;
    }
}
```

Naming Convention: Use Intention-revealing Name



```
public class Triangle {
    private double a;
    private double b;

    public Triangle(double a, double b){
        this.a = a;
        this.b = b;
    }

    public double A(){
        return 0.5 * a * b;
    }
}
```



```
public class Triangle {
    private double base;
    private double height;

    public Triangle(double base, double height){
        this.base = base;
        this.height = height;
    }

    public double calculateArea(){
        return 0.5 * base * height;
    }
}
```

Naming Convention: Avoid Disinformation

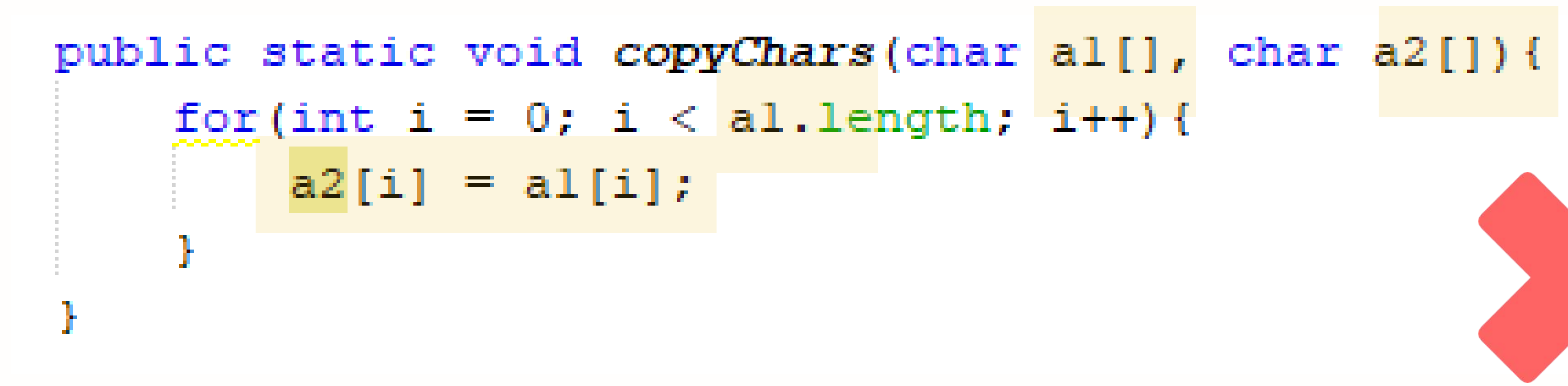
- We should avoid words whose entrenched meanings vary from our intended meaning
- Do not refer to a grouping of accounts as an `accountList` unless it's actually a `List`.
- Beware of using names which vary in small ways `XYZControllerForEfficientHandlingOfStrings` or `XYZControllerForEfficientStorageOfStrings`

Naming Convention: Make Meaningful Distinctions

- It is not sufficient to add number series or noise words, even though the compiler is satisfied.
  e.g.: creating a variable named `klass` just because the name `class` was used for something else

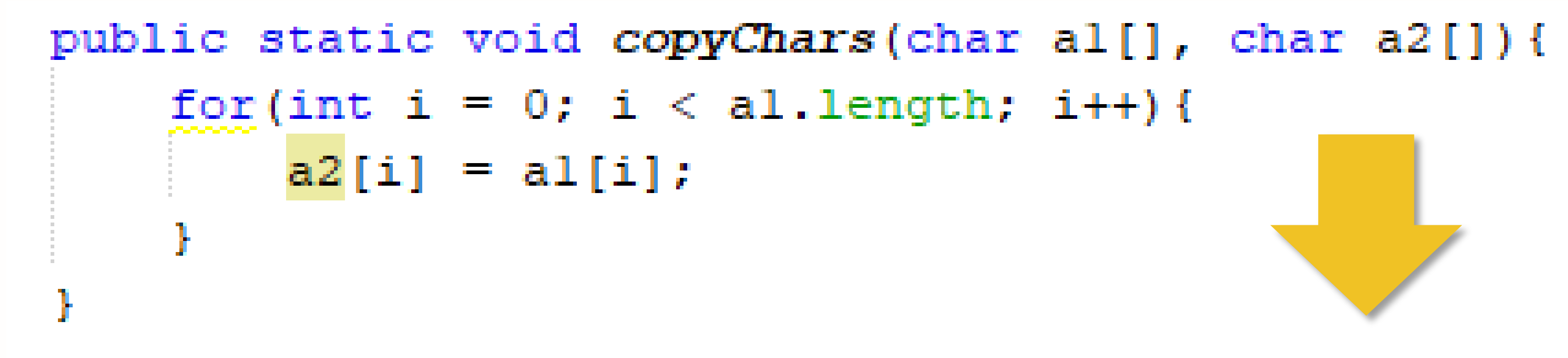
pens

Naming Convention: Make Meaningful Distinctions

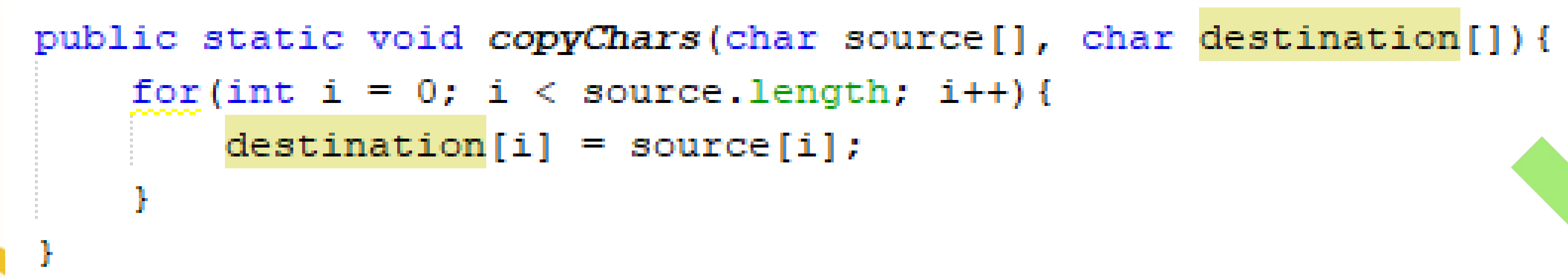


```
public static void copyChars(char a1[], char a2[]){
    for(int i = 0; i < a1.length; i++){
        a2[i] = a1[i];
    }
}
```

Naming Convention: Make Meaningful Distinctions

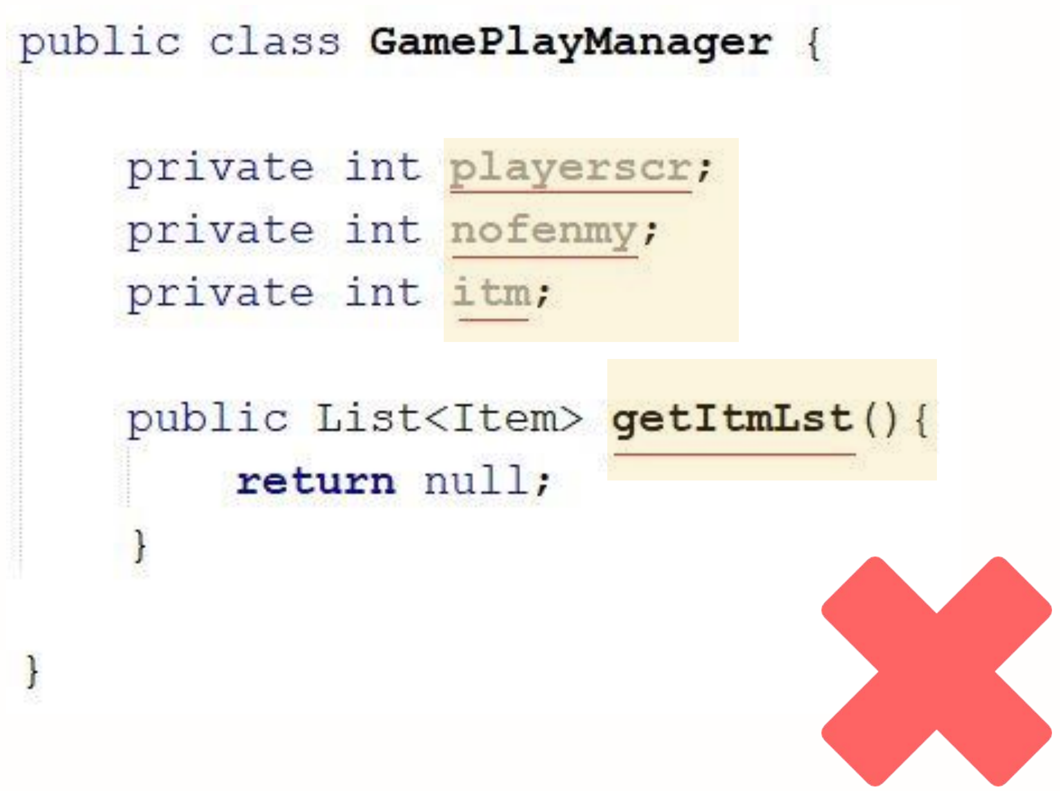


```
public static void copyChars(char a1[], char a2[]){
    for(int i = 0; i < a1.length; i++){
        a2[i] = a1[i];
    }
}
```



```
public static void copyChars(char source[], char destination[]){
    for(int i = 0; i < source.length; i++){
        destination[i] = source[i];
    }
}
```

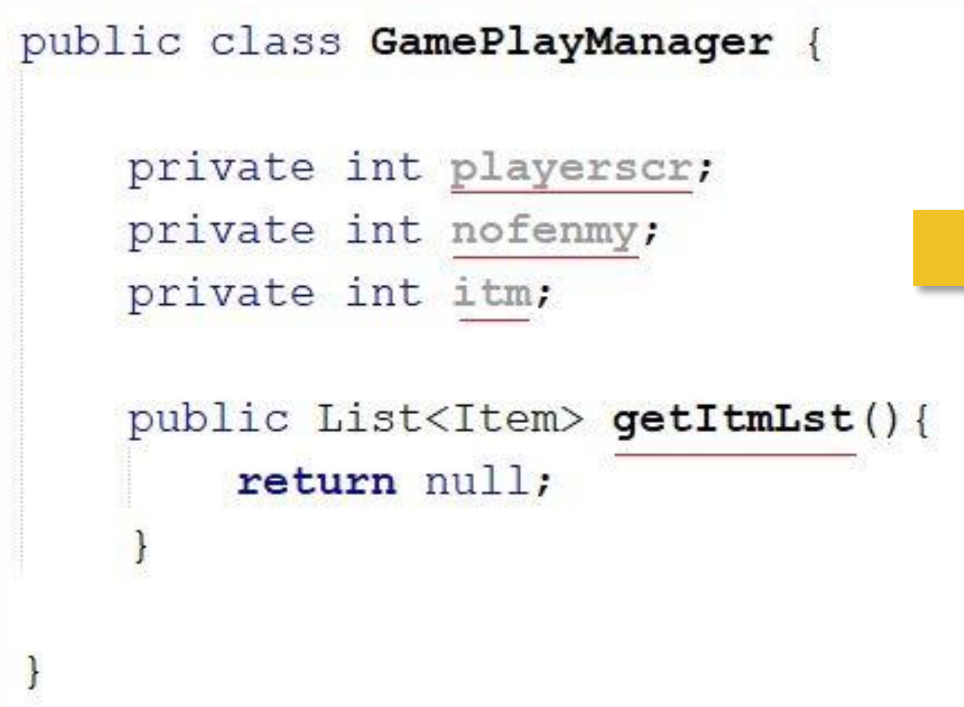
Naming Convention: Use Pronounceable Name



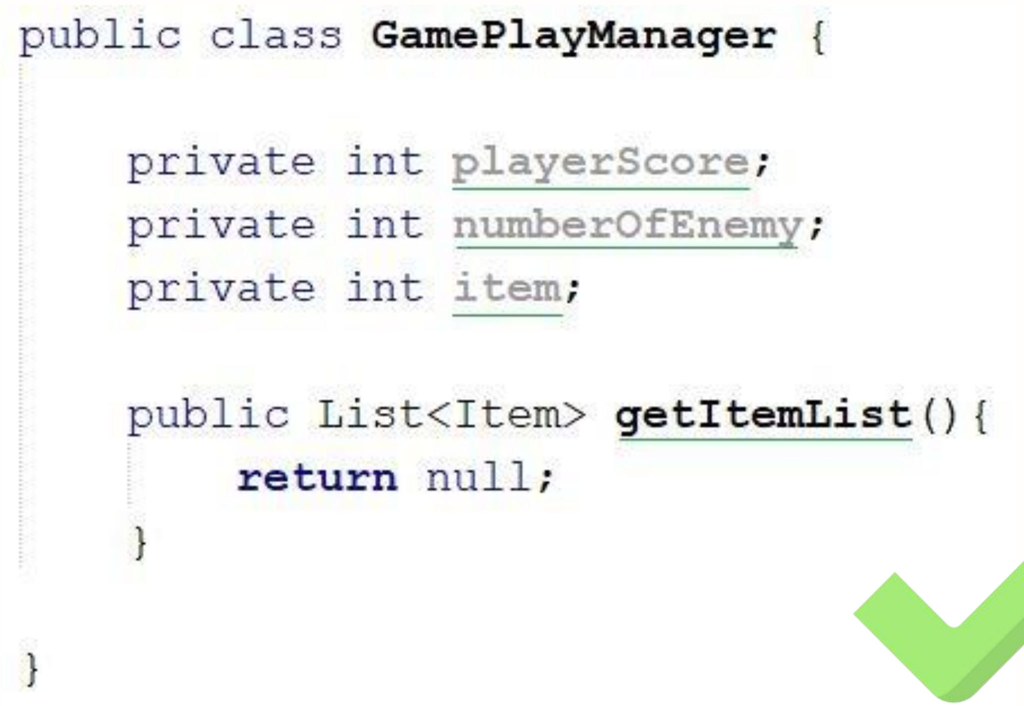
```
public class GamePlayManager {

    private int playerscr;
    private int nofenmy;
    private int itm;

    public List<Item> getItmLst(){
        return null;
    }

}
```

- If you can’t pronounce it, you can’t discuss it without sounding like an idiot

Naming Convention: Use Pronounceable Name

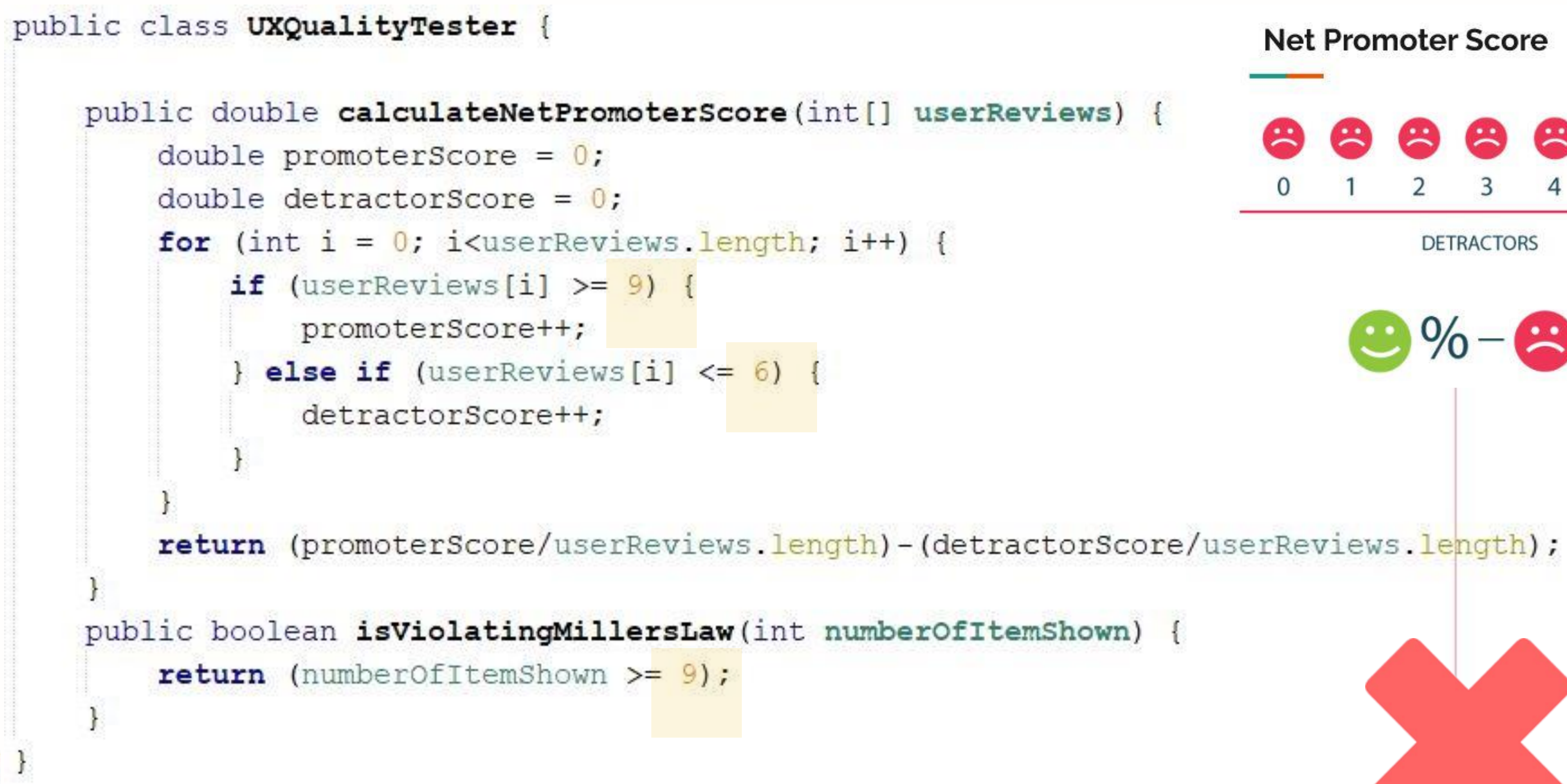


```
public class GamePlayManager {

    private int playerscr;
    private int nofenmy;
    private int itm;

    public List<Item> getItmLst(){
        return null;
    }

}
```

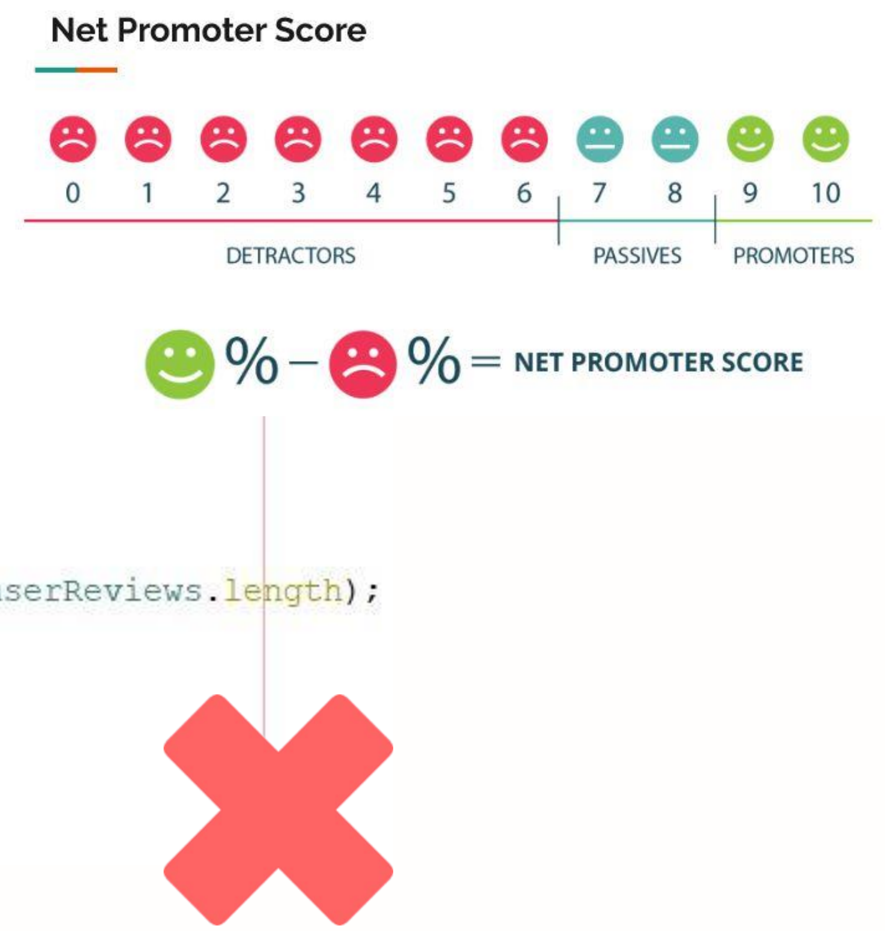


```
public class GamePlayManager {

    private int playerScore;
    private int numberOfEnemy;
    private int item;

    public List<Item> getItemList(){
        return null;
    }

}
```

Naming Convention: Use Searchable Name



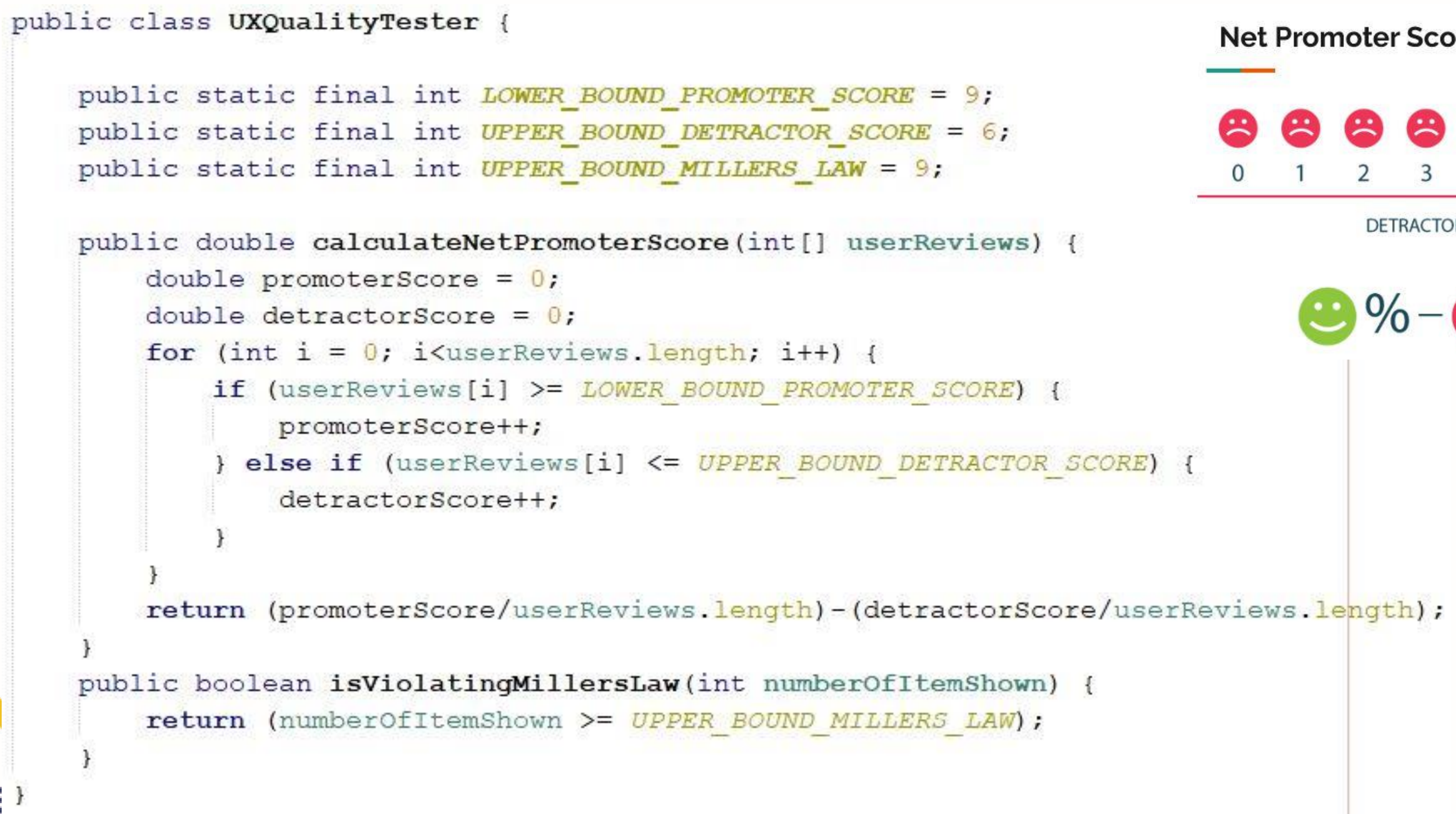
```
public class UXQualityTester {

    public double calculateNetPromoterScore(int[] userReviews) {
        double promoterScore = 0;
        double detractorScore = 0;
        for (int i = 0; i<userReviews.length; i++) {
            if (userReviews[i] >= 9) {
                promoterScore++;
            } else if (userReviews[i] <= 6) {
                detractorScore++;
            }
        }
        return (promoterScore/userReviews.length)-(detractorScore/userReviews.length);
    }
    public boolean isViolatingMillersLaw(int numberOfItemShown) {
        return (numberOfItemShown >= 9);
    }
}
```



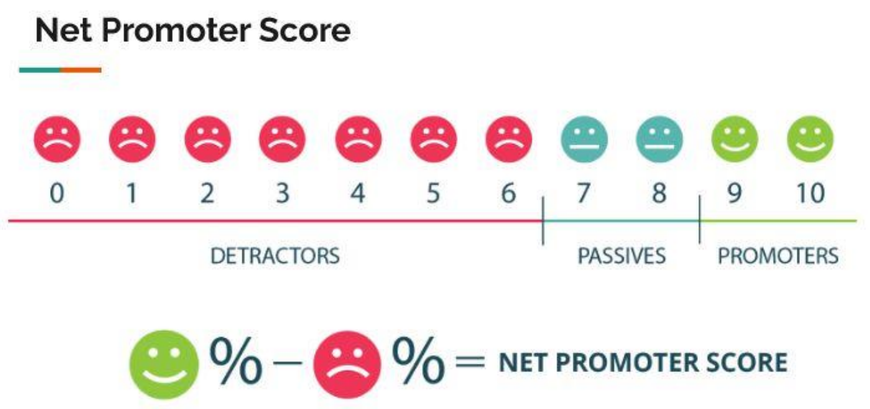
Naming Convention: Use Searchable Name

- Single-letter names and numeric constants have a particular problem in that they are not easy to locate across a body of text.
- Searches may turn up the digit as part of file names, other constant definitions, and in various expressions where the value is used with different intent.
- Single-letter names can ONLY be used as local variables inside short methods.

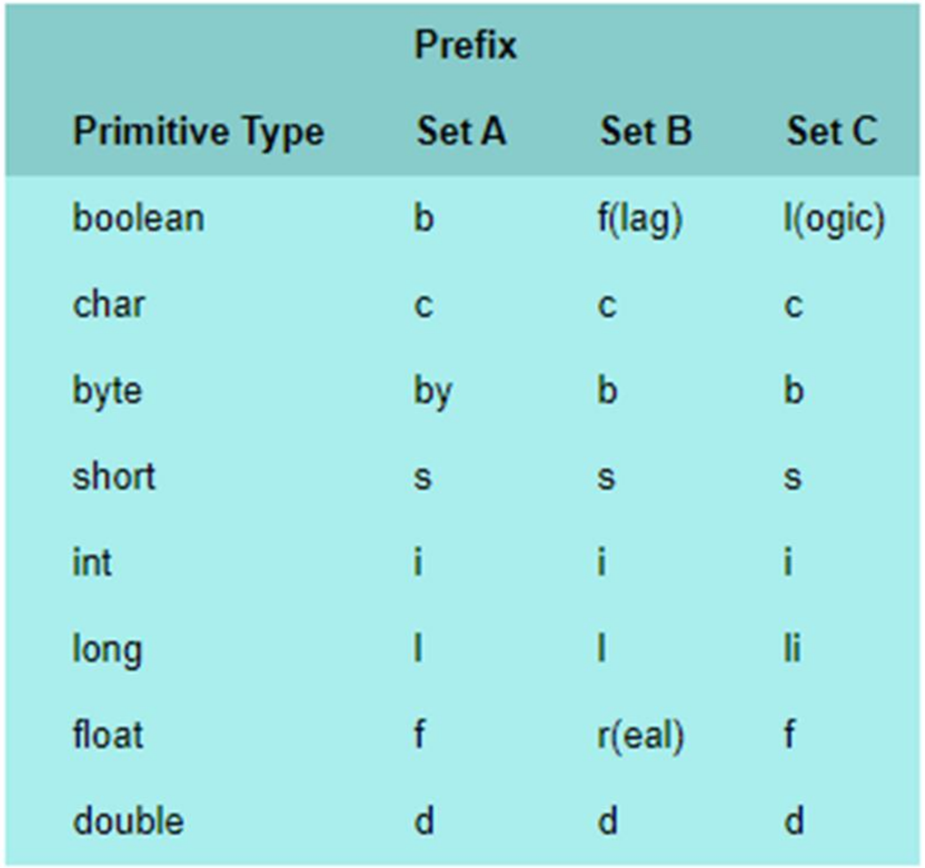
Naming Convention: Use Searchable Name



```
public class UXQualityTester {

    public static final int LOWER_BOUND_PROMOTER_SCORE = 9;
    public static final int UPPER_BOUND_DETRACTOR_SCORE = 6;
    public static final int UPPER_BOUND_MILLERS_LAW = 9;

    public double calculateNetPromoterScore(int[] userReviews) {
        double promoterScore = 0;
        double detractorScore = 0;
        for (int i = 0; i<userReviews.length; i++) {
            if (userReviews[i] >= LOWER_BOUND_PROMOTER_SCORE) {
                promoterScore++;
            } else if (userReviews[i] <= UPPER_BOUND_DETRACTOR_SCORE) {
                detractorScore++;
            }
        }
        return (promoterScore/userReviews.length)-(detractorScore/userReviews.length);
    }
    public boolean isViolatingMillersLaw(int numberOfItemShown) {
        return (numberOfItemShown >= UPPER_BOUND_MILLERS_LAW);
    }
}
```

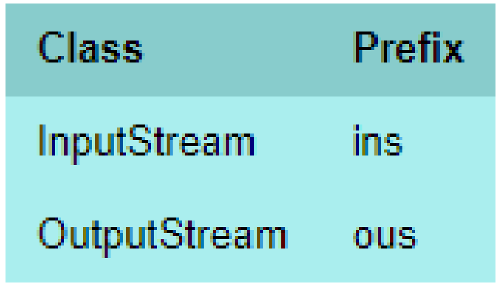


Naming Convention: Avoid Encoding (Hungarian Notation)



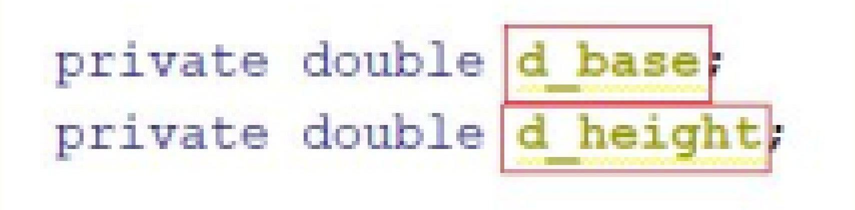
	Prefix		
Primitive Type	Set A	Set B	Set C
boolean	b	f(lag)	l(ogic)
char	c	c	c
byte	by	b	b
short	s	s	s
int	i	i	i
long	l	l	li
float	f	r(eal)	f
double	d	d	d

- Hungarian Notation (HN) is a naming convention that was originated years ago by Charles Simonyi of Microsoft.
- HN provides a way to make your code more understandable and maintainable by prefixing a variable name with its type.

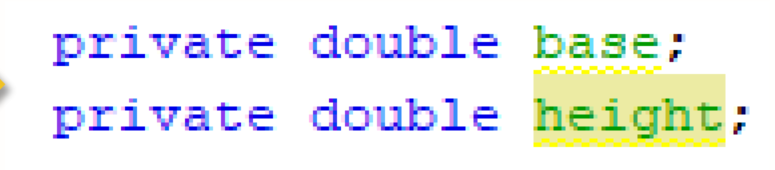


Class	Prefix
InputStream	ins
OutputStream	ous

Naming Convention: Avoid Encoding (Hungarian Notation)

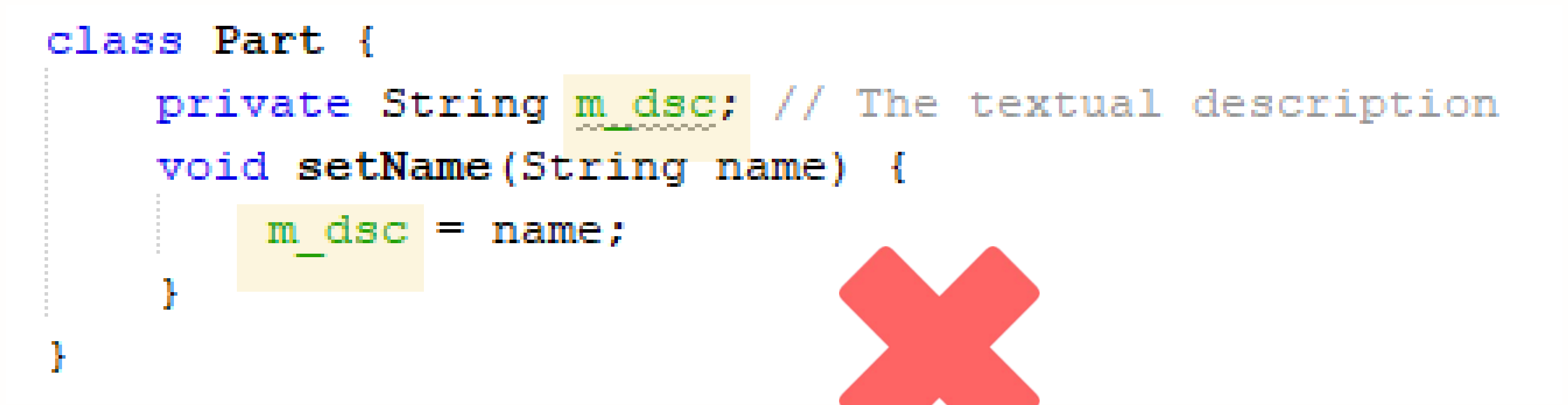


```
private double d_base;
private double d_height;
```



```
private double base;
private double height;
```

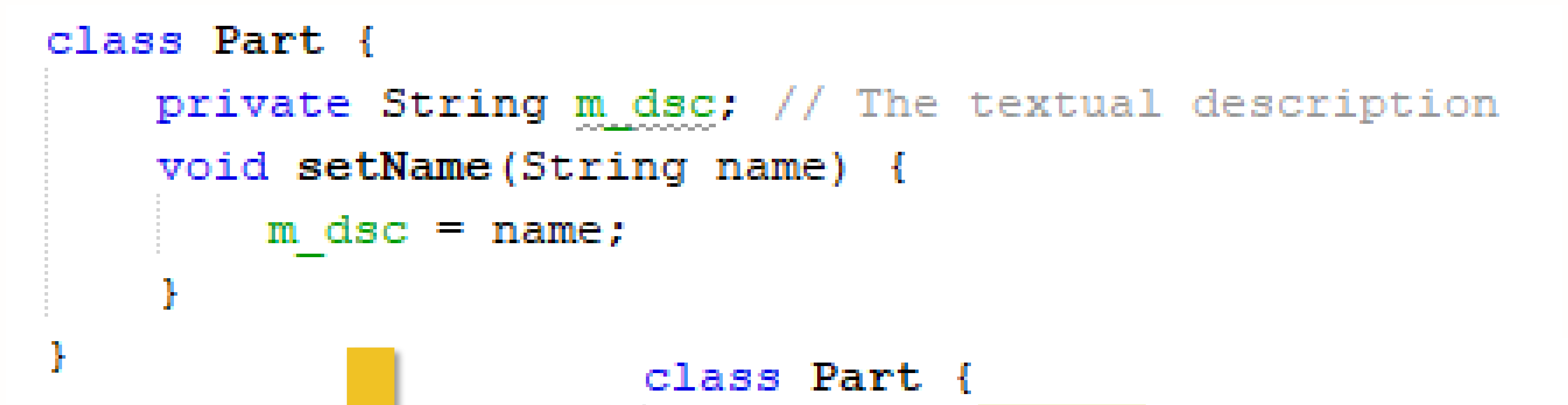
Naming Convention: Avoid Encoding (Member Prefix)



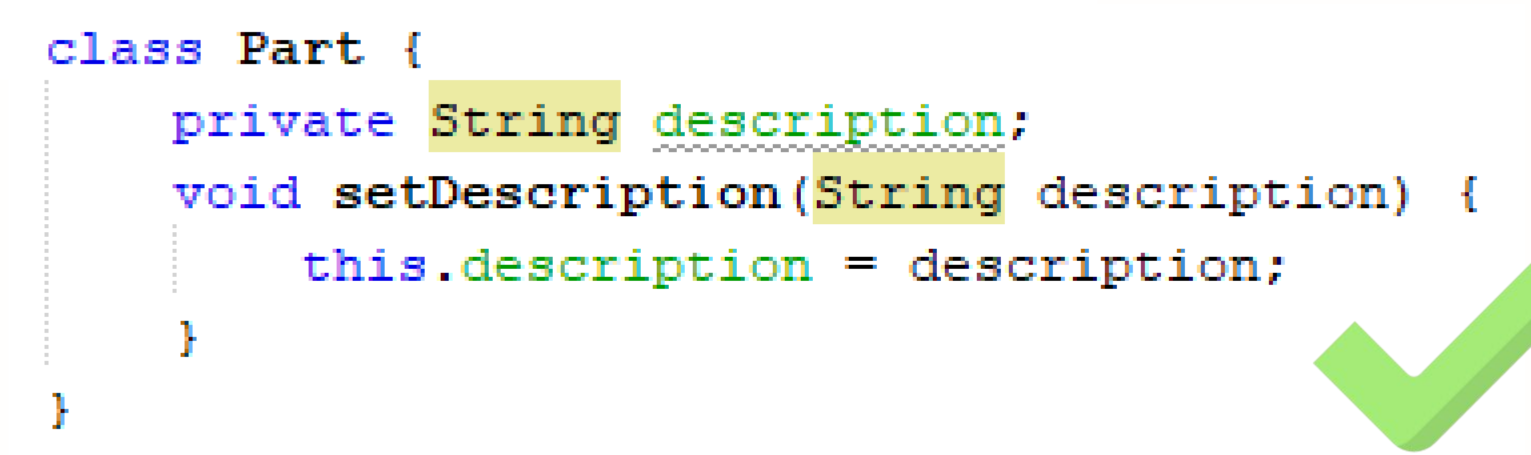
```
class Part {
    private String m_dsc; // The textual description
    void setName(String name) {
        m_dsc = name;
    }
}
```

✗

Naming Convention: Avoid Encoding (Member Prefix)



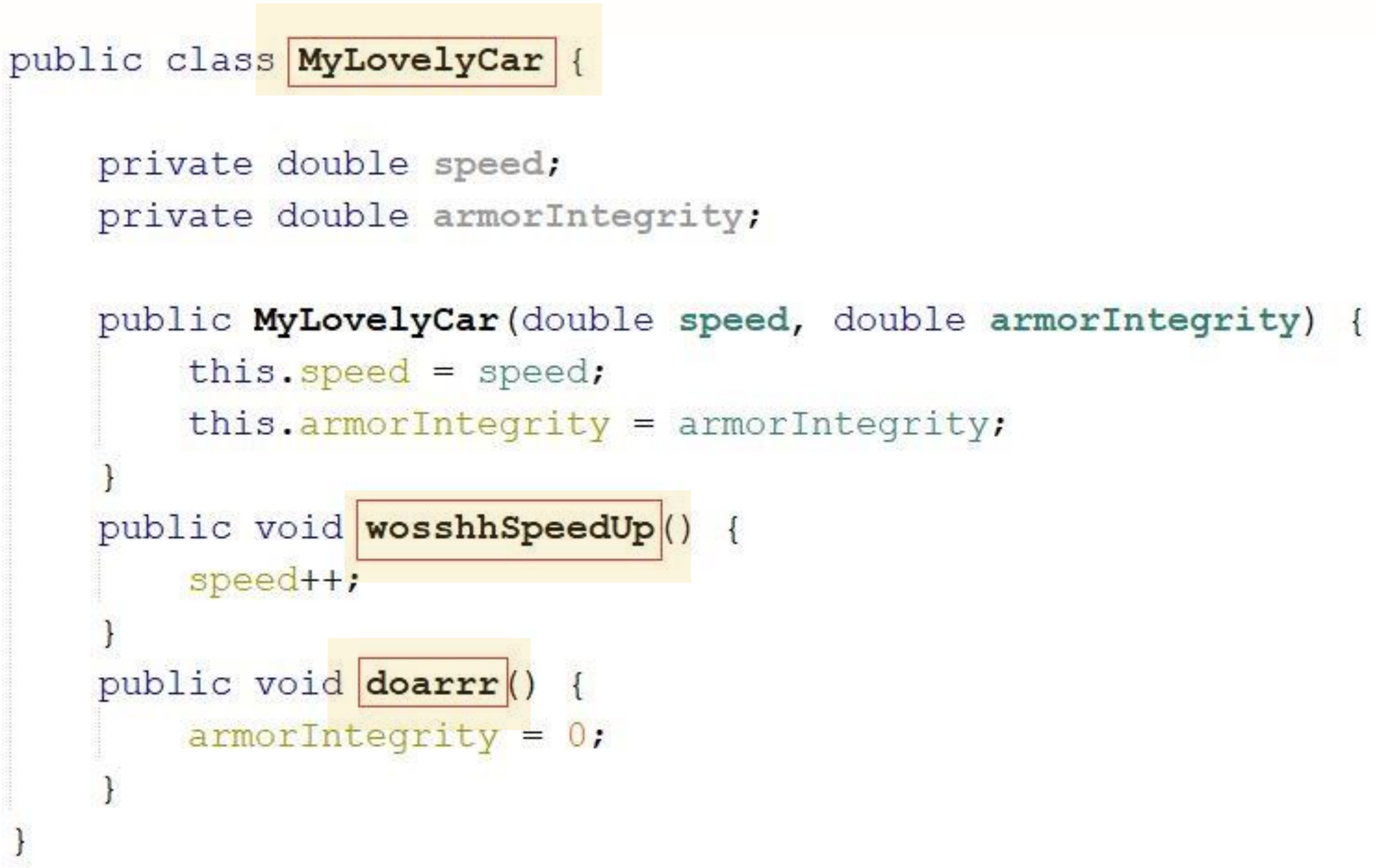
```java
class Part {
    private String m_dsc; // The textual description
    void setName(String name) {
        m_dsc = name;
    }
}
```



```java
class Part {
    private String description;
    void setDescription(String description) {
        this.description = description;
    }
}
```

Naming Convention: Do not be Cute

- Choose clarity over entertainment value.
- Cuteness in code often appears in the form of colloquialisms or slang.
- Say what you mean. Mean what you say.

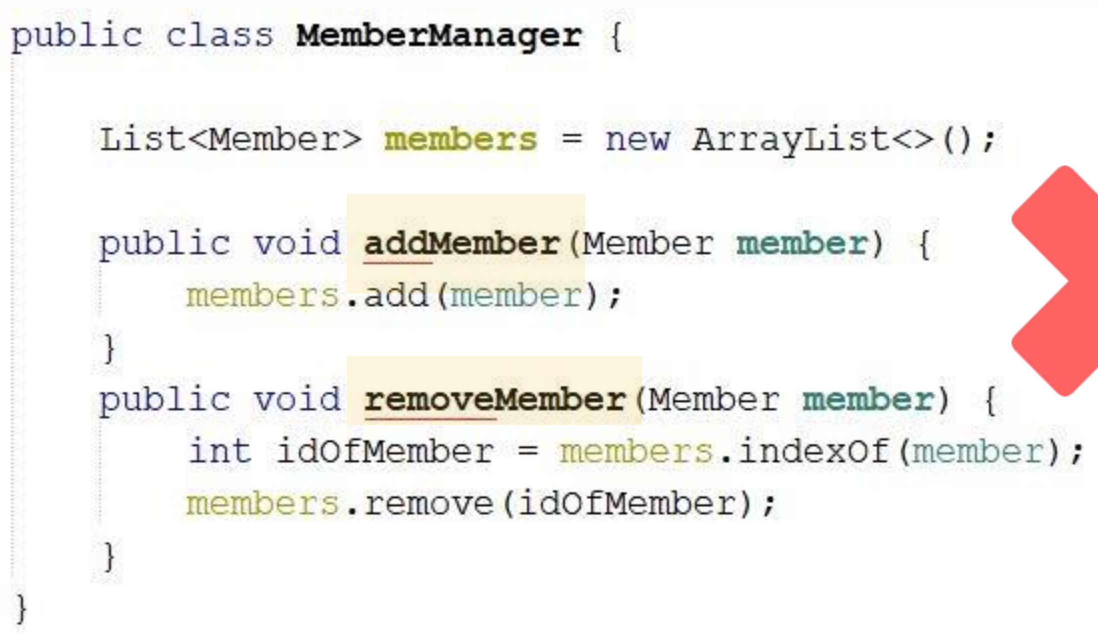


```
public class MyLovelyCar {

    private double speed;
    private double armorIntegrity;

    public MyLovelyCar(double speed, double armorIntegrity) {
        this.speed = speed;
        this.armorIntegrity = armorIntegrity;
    }
    public void wosshhSpeedUp() {
        speed++;
    }
    public void doarrr() {
        armorIntegrity = 0;
    }
}
```

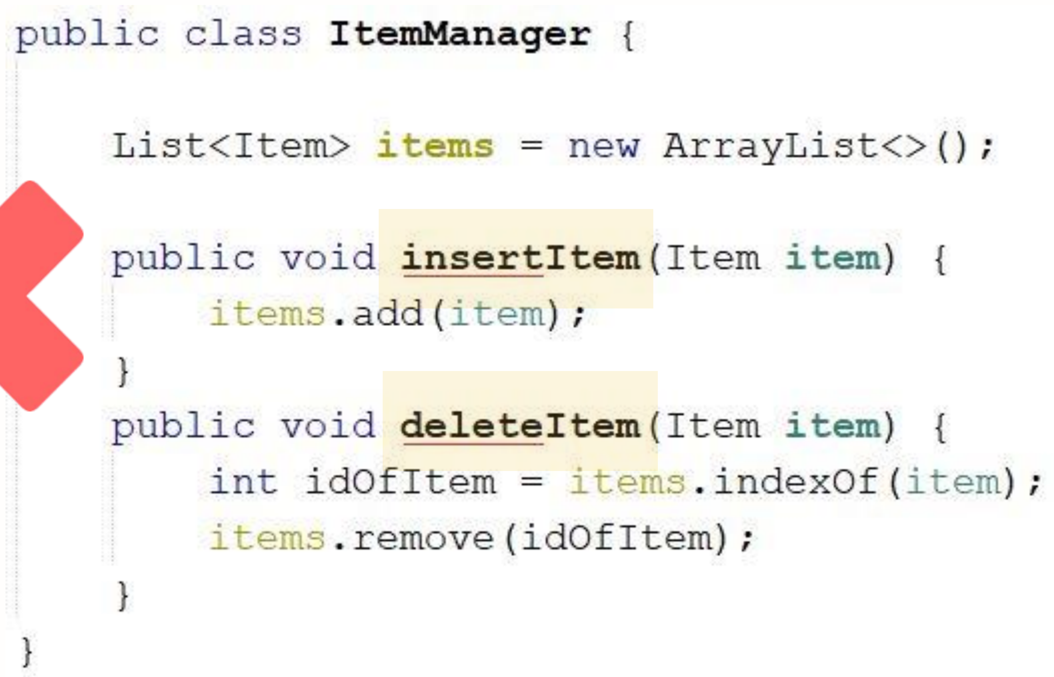
Naming Convention: Pick One Word per Concept

- Pick one word for one abstract concept and stick with it. e.g.: it's confusing to have `fetch()`, `retrieve()`, and `get()` as equivalent methods of different classes.
- A consistent lexicon is a great boon to the programmers who must use your code

Naming Convention: Pick One Word per Concept

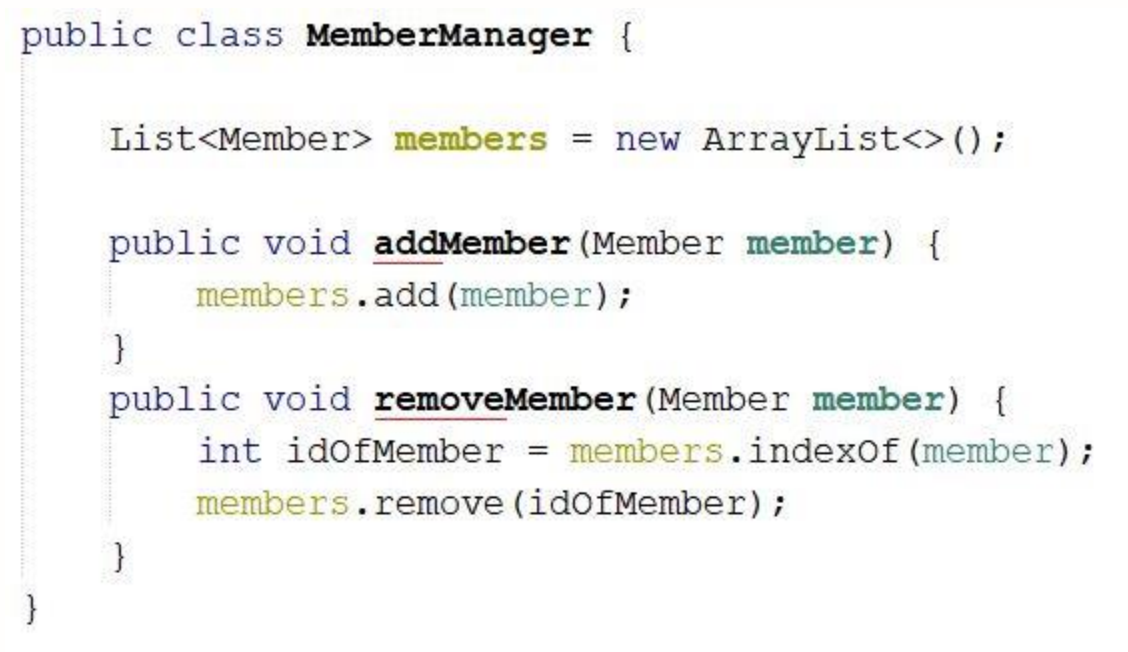


```
public class MemberManager {

    List<Member> members = new ArrayList<>();

    public void addMember(Member member) {
        members.add(member);
    }
    public void removeMember(Member member) {
        int idOfMember = members.indexOf(member);
        members.remove(idOfMember);
    }
}
```

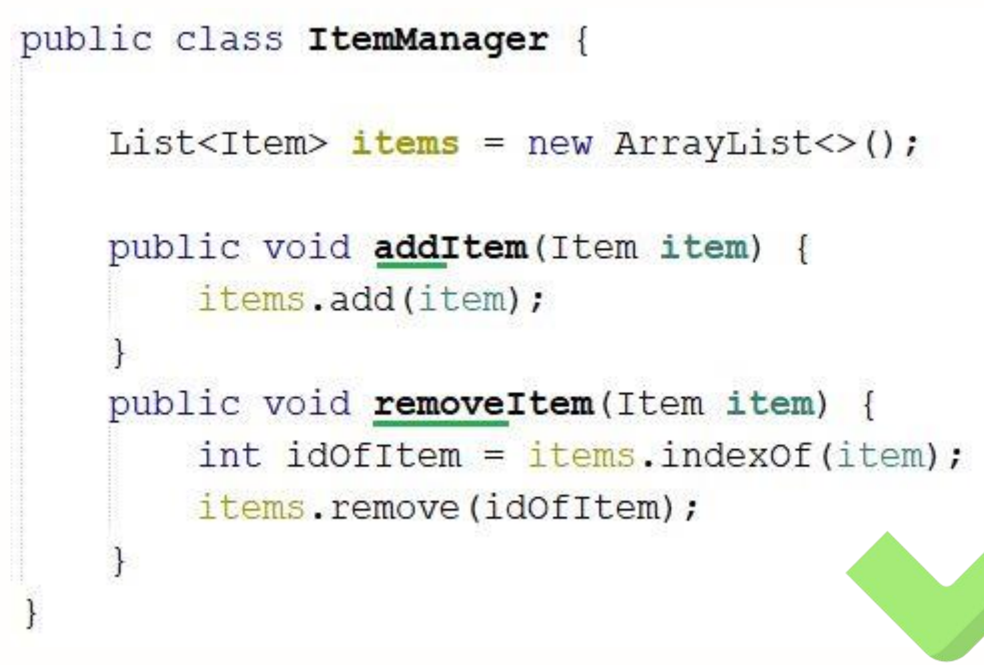


```
public class ItemManager {

    List<Item> items = new ArrayList<>();

    public void insertItem(Item item) {
        items.add(item);
    }
    public void deleteItem(Item item) {
        int idOfItem = items.indexOf(item);
        items.remove(idOfItem);
    }
}
```

Naming Convention: Pick One Word per Concept



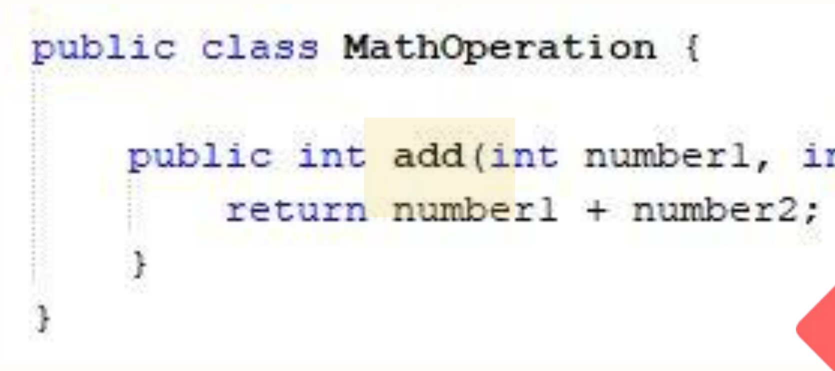
```
public class MemberManager {

    List<Member> members = new ArrayList<>();

    public void addMember(Member member) {
        members.add(member);
    }
    public void removeMember(Member member) {
        int idOfMember = members.indexOf(member);
        members.remove(idOfMember);
    }
}
```



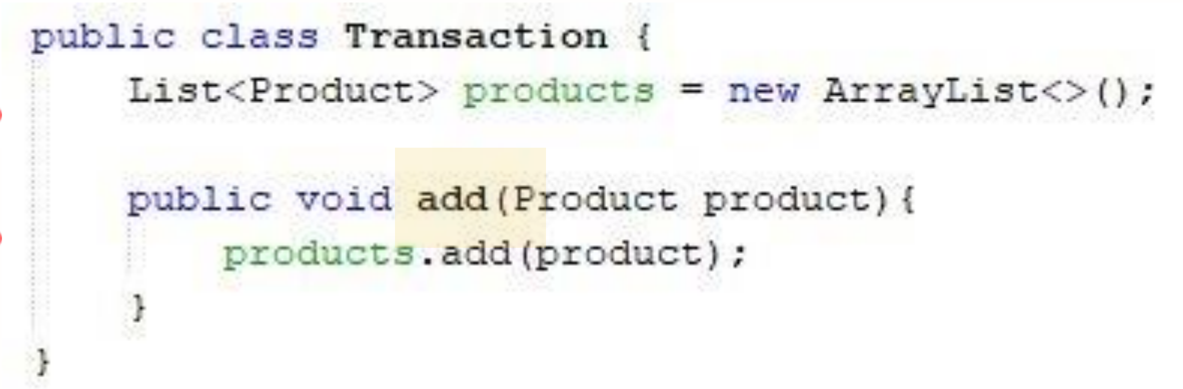
```
public class ItemManager {

    List<Item> items = new ArrayList<>();

    public void addItem(Item item) {
        items.add(item);
    }
    public void removeItem(Item item) {
        int idOfItem = items.indexOf(item);
        items.remove(idOfItem);
    }
}
```

Naming Convention: Do not Pun

- Avoid using the same word for two purposes.
  e.g: method add() can be diversified into insert() and append() according to its semantics.



```
public class MathOperation {

    public int add(int number1, int number2){
        return number1 + number2;
    }
}
```

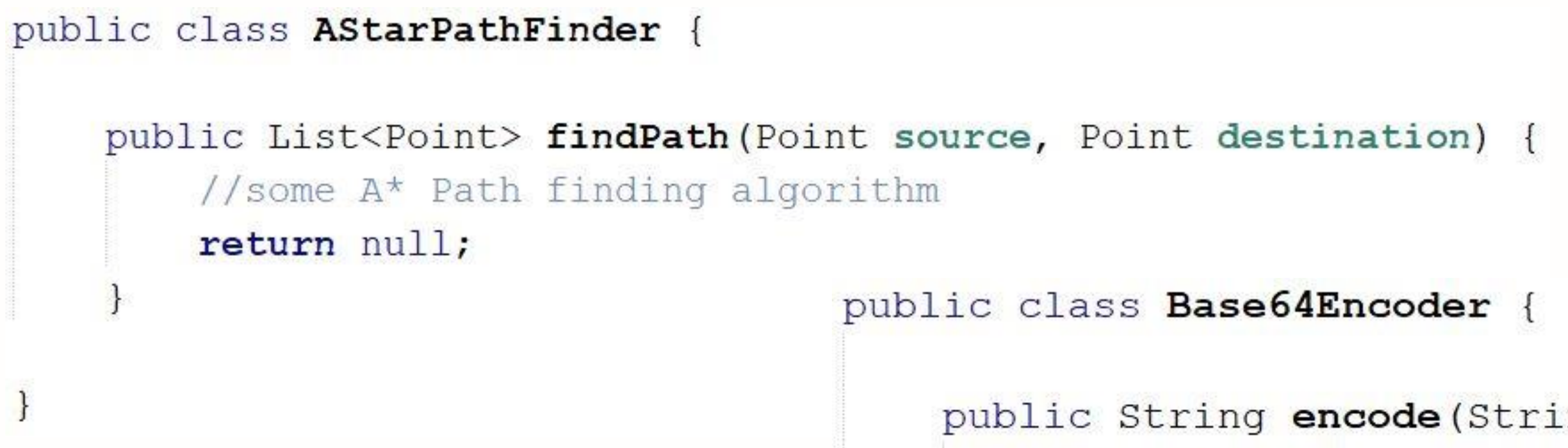


```
public class Transaction {
    List<Product> products = new ArrayList<>();

    public void add(Product product){
        products.add(product);
    }
}
```

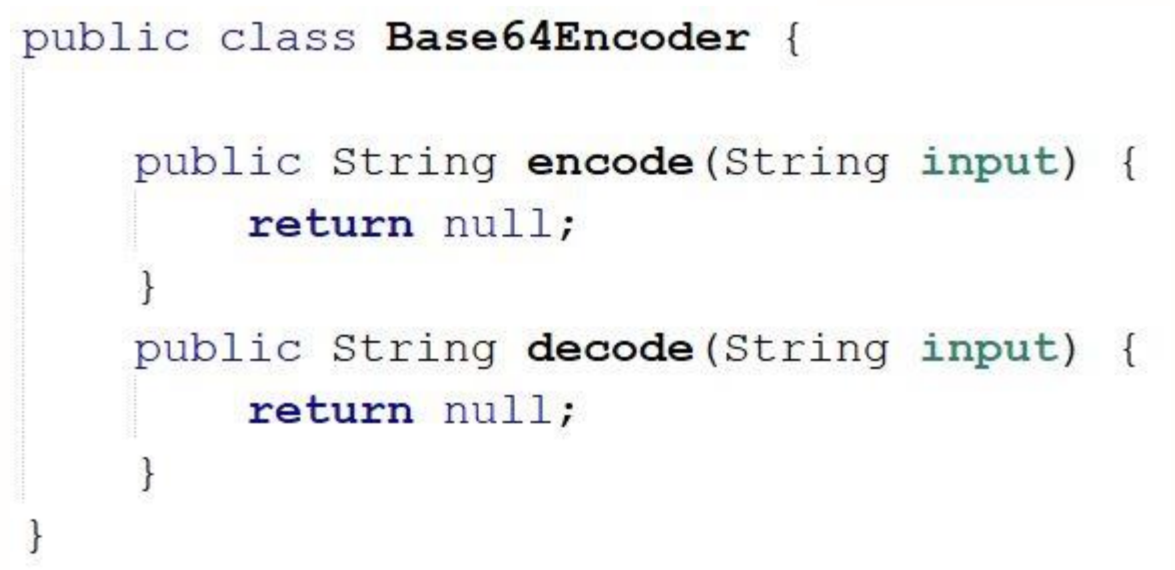
Naming Convention: Use Problem and Solution Domain Name

- Remember that the people who read your code will be programmers. So go ahead and use computer science (CS) terms, algorithm names, pattern names, math terms, and so forth ➔ solution domain name.
  e.g.:
  The name `AccountVisitor` means a great deal to a programmer who is familiar with the VISITOR pattern.
- When there is no "programmer-eese" for what you're doing, use the name from the problem domain. At least the programmer who maintains your code can ask a domain expert what it means.
  e.g.:
  `Equity` and `Liability` is common term in accounting.

Naming Convention: Use Problem and Solution Domain Name



```
public class AStarPathFinder {

    public List<Point> findPath(Point source, Point destination) {
        //some A* Path finding algorithm
        return null;
    }

}
```



```
public class Base64Encoder {

    public String encode(String input) {
        return null;
    }
    public String decode(String input) {
        return null;
    }
}
```

Example of
Solution Domain Name

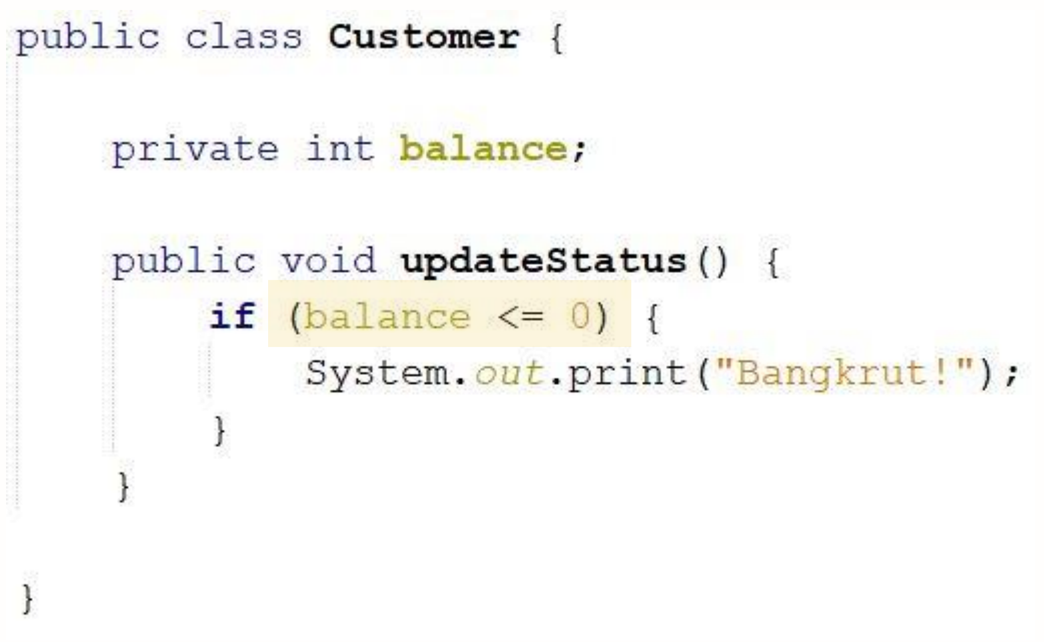
Naming Convention: Do not Add Gratuitous Context

- Shorter names are generally better than longer ones, so long as they are clear. Add no more context to a name than is necessary.
- In an imaginary application called “Gas Station Deluxe,” it is a bad idea to prefix every class with `GSD`.
- To differentiate among `MACAddresses`, `portAddresses`, and `WebAddresses`, it is better to use `MAC`, `port`, and `URI`.

Naming Convention: Add Meaningful Context

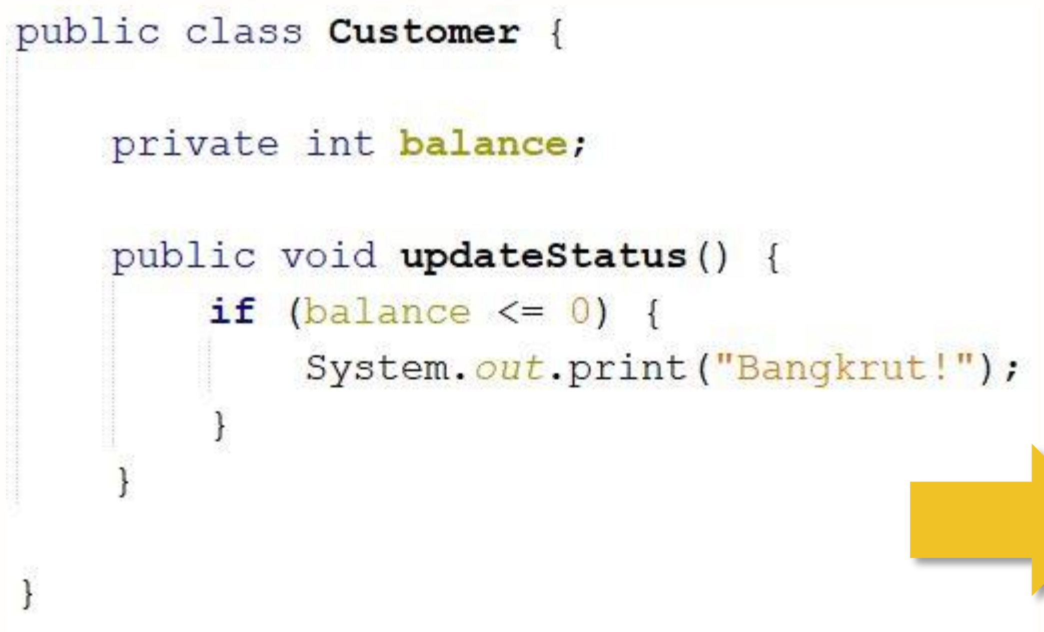
- There are a few names which are meaningful in and of themselves—most are not. Instead, you need to place names in context for your reader by enclosing them in well-named classes or functions.

Naming Convention: Add Meaningful Context

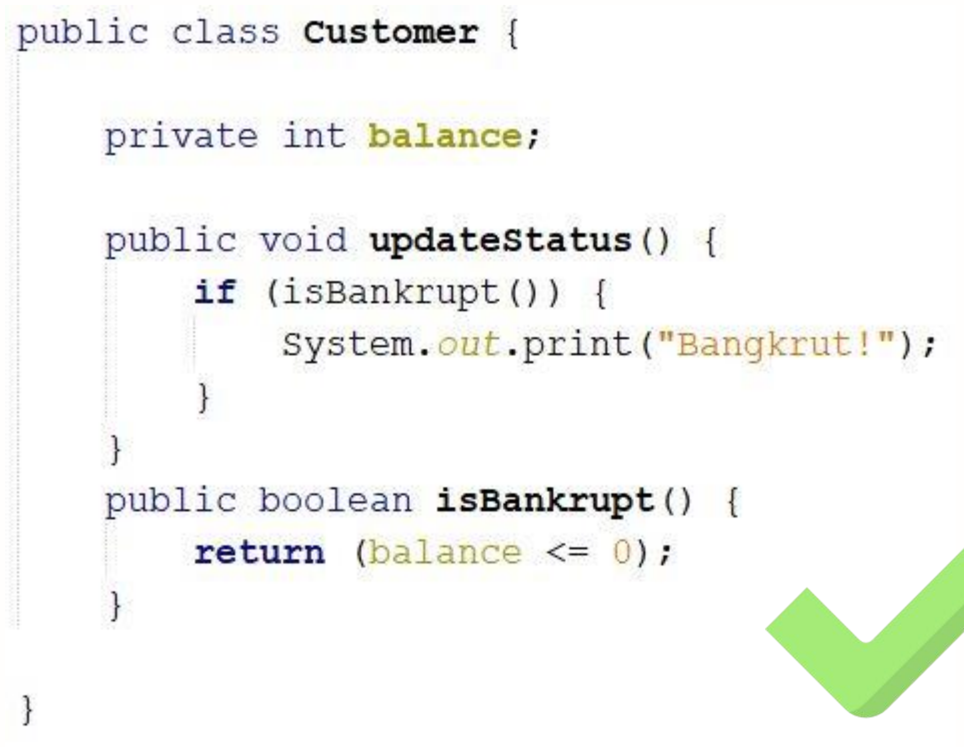


```
public class Customer {

    private int balance;

    public void updateStatus() {
        if (balance <= 0) {
            System.out.print("Bangkrut!");
        }
    }

}
```

Naming Convention: Add Meaningful Context

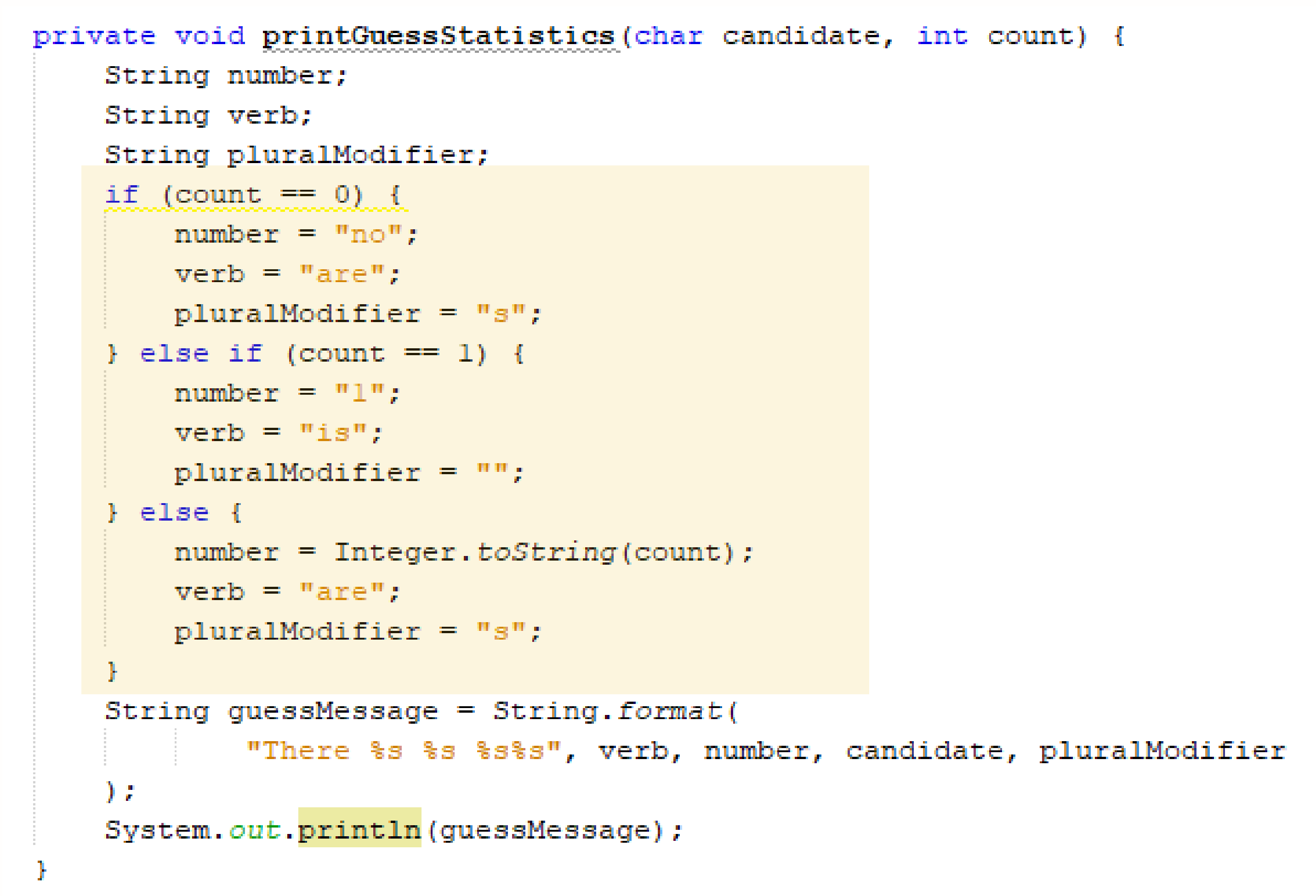


```java
public class Customer {

    private int balance;

    public void updateStatus() {
        if (balance <= 0) {
            System.out.print("Bangkrut!");
        }
    }

}
```



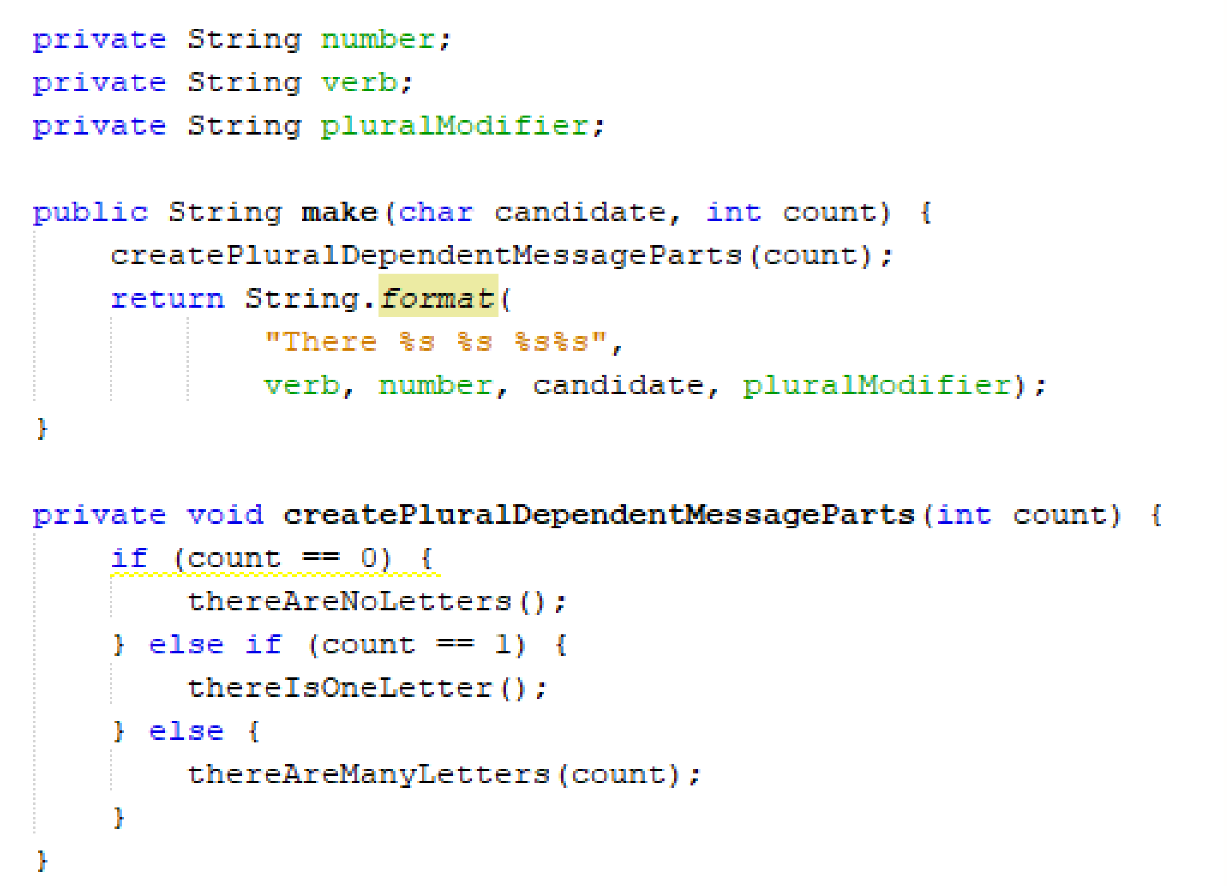
```java
public class Customer {

    private int balance;

    public void updateStatus() {
        if (isBankrupt()) {
            System.out.print("Bangkrut!");
        }
    }
    public boolean isBankrupt() {
        return (balance <= 0);
    }

}
```

Naming Convention: Add Meaningful Context

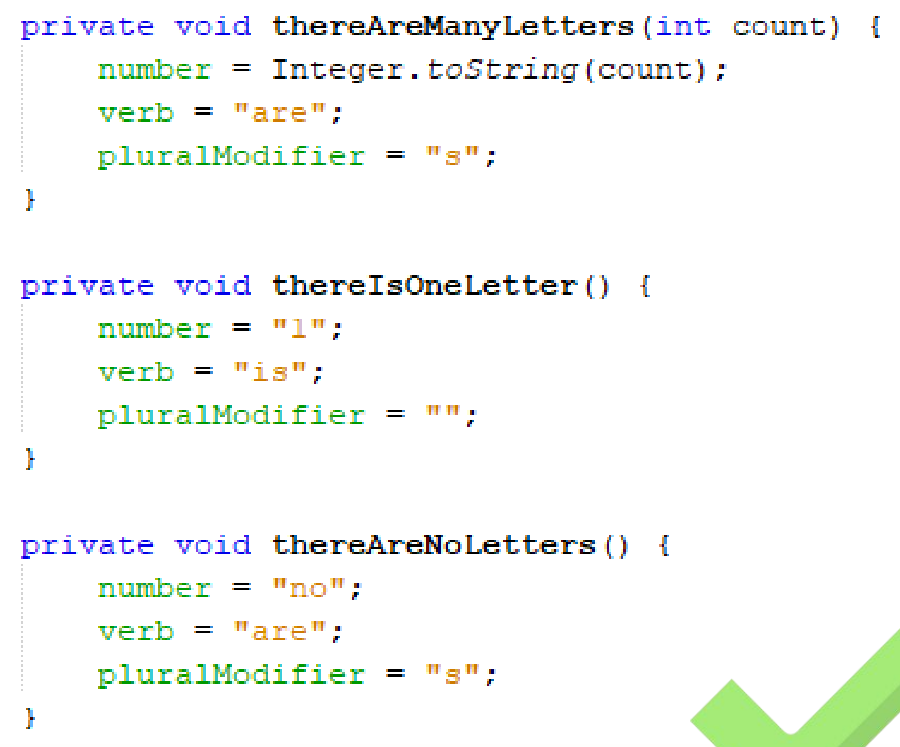


```java
private void printGuessStatistics(char candidate, int count) {
    String number;
    String verb;
    String pluralModifier;
    if (count == 0) {
        number = "no";
        verb = "are";
        pluralModifier = "s";
    } else if (count == 1) {
        number = "1";
        verb = "is";
        pluralModifier = "";
    } else {
        number = Integer.toString(count);
        verb = "are";
        pluralModifier = "s";
    }
    String guessMessage = String.format(
            "There %s %s %s%s", verb, number, candidate, pluralModifier
    );
    System.out.println(guessMessage);
}
```

Naming Convention: Add Meaningful Context



```java
private String number;
private String verb;
private String pluralModifier;

public String make(char candidate, int count) {
    createPluralDependentMessageParts(count);
    return String.format(
            "There %s %s %s%s",
            verb, number, candidate, pluralModifier);
}

private void createPluralDependentMessageParts(int count) {
    if (count == 0) {
        thereAreNoLetters();
    } else if (count == 1) {
        thereIsOneLetter();
    } else {
        thereAreManyLetters(count);
    }
}
```



```java
private void thereAreManyLetters(int count) {
    number = Integer.toString(count);
    verb = "are";
    pluralModifier = "s";
}

private void thereIsOneLetter() {
    number = "1";
    verb = "is";
    pluralModifier = "";
}

private void thereAreNoLetters() {
    number = "no";
    verb = "are";
    pluralModifier = "s";
}
```

References

- Rasyid Institute. Modul Workshop Clean Code. 2019.
- Martin, Robert C. Clean Code: A Handbook of Agile Software Craftsmanship. Pearson. 2008.

bridge to the future

http://www.eepis-its.edu